Process real-time big data with Twitter Storm

An introduction to streaming big data

M. Tim Jones
Independent author
Consultant

Skill Level: Intermediate

Date: 06 Nov 2012

Storm is an open source, big-data processing system that differs from other systems in that it's intended for distributed real-time processing and is language independent. Learn about Twitter Storm, its architecture, and the spectrum of batch and stream processing solutions.

Hadoop, the clear king of big-data analytics, is focused on batch processing. This model is sufficient for many cases (such as indexing the web), but other use models exist in which real-time information from highly dynamic sources is required. Solving this problem resulted in the introduction of Storm from Nathan Marz (now with Twitter by way of BackType). Storm operates not on static data but on streaming data that is expected to be continuous. With Twitter users generating 140 million tweets per day, it's easy to see how this technology is useful.

But Storm is more than a traditional big-data analytics system: It's an example of a complex event-processing (CEP) system. CEP systems are typically categorized as computation and detection oriented, each of which can be implemented in Storm through user-defined algorithms. CEPs can, for example, be used to identify meaningful events from a flood of events, and then take actions on those events in real time.

Nathan Marz provides a number of examples in which Storm is used within Twitter. One of the most interesting is the generation of trend information. Twitter extracts emerging trends from the fire hose of tweets and maintains them at the local and national level. What this means is that as a story begins to emerge, Twitter's trending topics algorithm identifies the topic in real time. This real-time algorithm is implemented within Storm as a continuous analysis of Twitter data.

What does "big data" mean?

Big data refers to a scale of data that cannot be managed by conventional means. Internet-scale data has fostered the creation of new architectures

© Copyright IBM Corporation 2012

and applications that are able to process this new class of data. These architectures are highly scalable and efficiently process data in parallel across a sea of servers.

Storm versus traditional big data

What differentiates Storm from other big-data solutions is the paradigm that it addresses. Hadoop is fundamentally a batch processing system. Data is introduced into the Hadoop file system (HDFS) and distributed across nodes for processing. When the processing is complete, the resulting data is returned to HDFS for use by the originator. Storm supports the construction of topologies that transform unterminated streams of data. Those transformations, unlike Hadoop jobs, never stop, instead continuing to process data as it arrives.

Big data implementations

Hadoop's core was written in the Java™ language but supports data analytics applications written in a variety of languages. Recent entrants have gone more esoteric routes for their implementations to exploit modern languages and their features. For example, the University of California (UC), Berkeley's, Spark is implemented in the Scala language, while Twitter Storm is implemented in Clojure (pronounced *closure*).

Clojure is a modern dialect of the Lisp language. Clojure, like Lisp, supports a functional style of programming, but Clojure also incorporates features to simplify multithreaded programming (a useful feature for the construction of Storm). Clojure is a virtual machine (VM)-based language that runs on the Java Virtual Machine. But even though Storm was developed in Clojure, you can write applications within Storm in virtually any language. All that's necessary is an adapter to connect to Storm's architecture. Adapters exist for Scala, JRuby, Perl, and PHP, and there's a Structured Query Language adapter that supports streaming into a Storm topology.

Key attributes of Storm

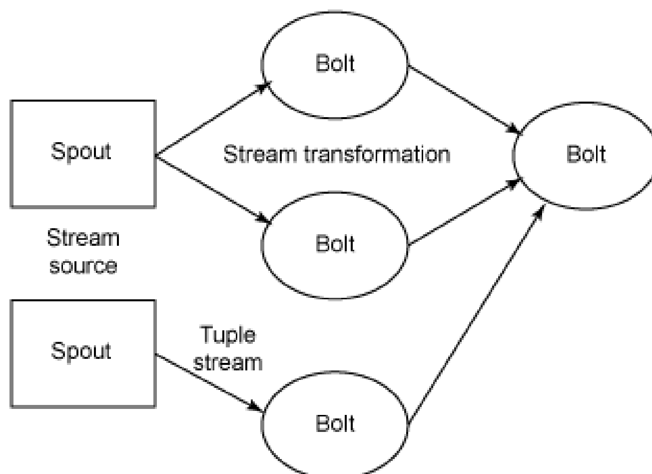
Storm implements a set of characteristics that define it in terms of performance and reliability. Storm uses ZeroMQ for message passing, which removes intermediate queueing and allows messages to flow directly between the tasks themselves. Under the covers of messaging is an automated and efficient mechanism for serialization and deserialization to Storm's primitive types.

What makes Storm most interesting is its focus on fault tolerance and management. Storm implements guaranteed message processing such that each tuple is fully processed through the topology; if a tuple is discovered not to have been processed, it is automatically replayed from the spout. Storm also implements fault detection at the task level, where upon failure of a task, messages are automatically reassigned to quickly restart processing. Storm includes more intelligent process management than Hadoop, where processes are managed by supervisors to ensure that resources are adequately used.

The Storm model

Storm implements a data flow model in which data flows continuously through a network of transformation entities (see Figure 1). The abstraction for a data flow is called a *stream,* which is an unbounded sequence of tuples. The tuple is like a structure that can represent standard data types (such as ints, floats, and byte arrays) or user-defined types with some additional serialization code. Each stream is defined by a unique ID that can be used to build topologies of data sources and sinks. Streams originate from *spouts,* which flow data from external sources into the Storm topology.

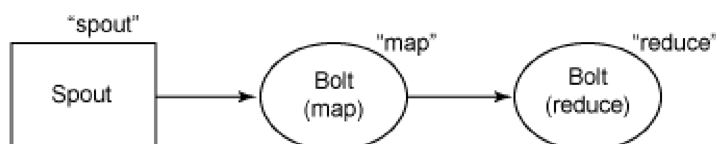
Figure 1. Conceptual architecture of a trivial Storm topology



The *sinks* (or entities that provide transformations) are called *bolts.* Bolts implement a single transformation on a stream and all processing within a Storm topology. Bolts can implement traditional things like MapReduce functionality or more complex actions (single-step functions) like filtering, aggregations, or communication with external entities such as a database. A typical Storm topology implements multiple transformations and therefore requires multiple bolts with independent tuple streams. Both spouts and bolts are implemented as one or more tasks within a Linux® system.

You can use Storm to easily implement MapReduce functionality for word frequency. As shown in Figure 2, a spout generates the stream of textual data, and a bolt implements the Map function (to tokenize the words of a stream). The resulting stream from the "map" bolt then flows into a single bolt that implements the Reduce function (to aggregate the words into counts).

Figure 2. Simple Storm topology for the MapReduce function



Note that bolts can stream data to multiple bolts as well as accept data from multiple sources. Storm has the concept of *stream groupings,* which implement shuffling

(random but equal distribution of tuples to bolts) or field grouping (stream partitioning based upon the fields of the stream). Other stream groupings exist, including the ability for the producer to route tuples using its own internal logic.

But one of the most interesting features in the Storm architecture is the concept of *guaranteed message processing.* Storm can guarantee that every tuple a spout emits will be processed; if it isn't processed within some timeout, Storm replays the tuple from the spout. This functionality requires some clever tricks for tracking the tuple through the topology and is one of Storm's key value adds.

In addition to supporting reliable messaging, Storm uses ZeroMQ to maximize messaging performance (removing intermediate queueing and implementing direct passing of messages between tasks). ZeroMQ incorporates congestion detection and alters its communication to optimize the available bandwidth.

Storm through example

Let's now look at an example of Storm through code to implement a simple MapReduce topology (see Listing 1). This example uses the nicely constructed word count example from Nathan's storm-starter kit available from GitHub (see Resources for a link). This example illustrates the topology shown in Figure 2, which implements a map transformation consisting of a bolt and a reduce transformation consisting of a single bolt.

Listing 1. Build a topology in Storm for Figure 2

```
01  TopologyBuilder builder = new TopologyBuilder();
02
03  builder.setSpout("spout", new RandomSentenceSpout(), 5);
04
05  builder.setBolt("map", new SplitSentence(), 4)
06           .shuffleGrouping("spout");
07
08  builder.setBolt("reduce", new WordCount(), 8)
09           .fieldsGrouping("map", new Fields("word"));
10
11  Config conf = new Config();
12  conf.setDebug(true);
13
14  LocalCluster cluster = new LocalCluster();
15  cluster.submitTopology("word-count", conf, builder.createTopology());
16
17  Thread.sleep(10000);
18
19  cluster.shutdown();
```

Listing 1 (line numbers added for reference) begins with the declaration of a new topology using the `TopologyBuilder`. Next, at line 3, a spout is defined (named `spout`) that consists of a `RandomSentenceSpout`. The `RandomSentenceSpout` class (namely, the `nextTuple` method) emits one of five random sentences as its data. The `5` argument at the end of the `setSpout` method is a parallelism hint (or the number of tasks to create for this activity).

At lines 5 and 6, I define the first bolt (or algorithmic transformation entity)—in this case, the map (or split) bolt. This bolt uses the `SplitSentence` to tokenize the input stream and emit as its output individual words. Note the use of `shuffleGrouping` at line 6, which defines the input subscription for this bolt (in this case, the "spout") but also that the stream grouping is defined as shuffle. This shuffle grouping means that the input from the spout will be *shuffled,* or randomly distributed to tasks within this bolt (which has a hint of four-task parallelism).

At lines 8 and 9, I define the last bolt, which effectively serves as the reduce element, with its input as the map bolt. The `WordCount` method implements the necessary word counting behavior (grouping like words together to maintain an overall count) but is not shuffled, so that its output is consistent. If there were multiple tasks implementing the reduce behavior, you would end up with segmented counts, not overall counts.

Lines 11 and 12 create and define a configuration object and enable Debug mode. The `Config` class contains a large number of configuration possibilities (see Resources for a link to more information on the Storm class tree).

Lines 14 and 15 create the local cluster (in this case, defining the use of Local mode). I define the name of my local cluster, my configuration object, and my topology (retrieved through the `createTopology` element of the `builder` class).

Finally, at line 17, Storm sleeps for a duration, and then shuts down the cluster at line 19. Remember that Storm is a continuously operating system, so tasks can exist for a considerable amount of time, operating on new tuples in streams for which they're subscribed.

You can learn more about this surprisingly simple implementation, including the details of the spout and bolts, in the storm-starter kit.

Using Storm

Nathan Marz has written a readable set of documentation that details installing Storm for both cluster and local modes of operation. The Local mode allows use of Storm without the requirement of a large cluster of nodes. If you need to use Storm in a cluster but lack the nodes, you can also implement a Storm cluster in Amazon Elastic Compute Cloud (EC2). See Resources for a reference for each Storm mode (Local, Cluster, and Amazon EC2).

Other open source big-data solutions

Since Google introduced the MapReduce paradigm in 2004, several solutions have appeared that use (or have qualities) of the original MapReduce paradigm. Google's original application of MapReduce was for the indexing of the World Wide Web. Although this application remains in popular usage, the problems that this simple model solves are growing.

Table 1 provides a list of the available open source big-data solutions, including traditional batch and streaming applications. Nearly a year before Storm was introduced to open source, Yahoo!'s S4 distributed stream computing platform was open sourced to Apache. S4 was released in October 2010 and provides a high-performance computing (HPC) platform that hides the complexity of parallel processing from the application developer. S4 implements a decentralized cluster architecture that is scalable and incorporates partial fault tolerance.

Table 1. Open source big-data solutions

Solution	Developer	Type	Description
Storm	Twitter	Streaming	Twitter's new streaming big-data analytics solution
S4	Yahoo!	Streaming	Distributed stream computing platform from Yahoo!
Hadoop	Apache	Batch	First open source implementation of the MapReduce paradigm
Spark	UC Berkeley AMPLab	Batch	Recent analytics platform that supports in-memory data sets and resiliency
Disco	Nokia	Batch	Nokia's distributed MapReduce framework
HPCC	LexisNexis	Batch	HPC cluster for big data

Going forward

Although Hadoop continues to be the most publicized big-data analytics solution, numerous other possibilities exist, each with different characteristics. I have explored Spark in past articles, which incorporates in-memory capabilities for data sets (with the ability to rebuild data that has been lost). But both Hadoop and Spark focus on the batch processing of large data sets. Storm provides a new model for big-data analytics and, because it was recently open sourced, has generated a considerable amount of interest.

Unlike Hadoop, Storm is a computation system and incorporates no concept of storage. This allows Storm to be used in a variety of contexts, whether data arrives dynamically from a nontraditional source or is stored in a storage system such as a database (or consumed by a controller for real-time manipulation of some other device, such as a trading system).

See Resources for links to more information on Storm, how to get a cluster up and running, and other big-data analytics solutions (both batch and streaming).

Resources

Learn

- Complex event processing is the pattern implemented by Storm as well as many other solutions, like Yahoo!' S4. A key difference between Storm and S4 is that Storm provides guaranteed message processing in the face of failures, where S4 can lose messages.
- Nathan Marz, the key developer behind Storm, has written several interesting and useful introductions to his new offering. The earliest glimpse of Storm came in May 2011 in Preview of Storm: The Hadoop of Realtime Processing - BackType Technology, which was followed in August by A Storm is coming: more details and plans for release.
- The Storm wiki provides a great set of documentation on Storm, its rationale, and a variety of tutorials on getting Storm and setting up a new project. You'll also find a useful set of documentation on many aspects of Storm, including use of Storm in Local mode, in clusters, and on Amazon.
- Spark, an alternative for fast data analytics (M. Tim Jones, developerWorks, November 2011) provides an introduction to UC Berkeley's in-memory resilient data analytics platform.
- Application virtualization, past and future (M. Tim Jones, developerWorks, May 2011) details the use of virtualization for language abstractions. Storm uses the VM-based language Clojure for its implementation in addition to Java technology and many other languages to build its internal (bolt) applications.
- A thorough class tree exists at GitHub for Storm that details its classes and interfaces.
- Hadoop has begun to address models beyond simple batch processing. For example, through scheduling, Hadoop can alter the way it processes data to focus on interactivity over batch-level data processing. Learn more about Hadoop scheduling in Scheduling in Hadoop (M. Tim Jones, developerWorks, December 2011).
- The Open Source developerWorks zone provides a wealth of information on open source tools and using open source technologies.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools, as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Follow developerWorks on Twitter. You can also follow this author on Twitter at M. Tim Jones.

Get products and technologies

- ZeroMQ is the intelligent transport layer for efficient messaging in scalable environments. At the ZeroMQ site, you can learn about the offering, how to use it to solve problems, and also how to support this effort.
- Apache Zookeeper is an open source project that enables highly reliable distributed coordination. Storm uses Zookeeper to coordinate amongst a set of nodes within a cluster.
- Clojure is the language used to implement the Storm system. Clojure is a recent derivative of the Lisp language created by Rich Hicky as a general-purpose language that also simplifies multithreaded programming.
- Apache Hadoop is the platform developed by Yahoo! for MapReduce programming. It was recently followed by Spark from UC Berkeley as a resilient in-memory, open source, big-data offering developed in Scala.
- In addition to Storm, several other big-data offerings are available as open source. Yahoo! S4 is another stream-based big-data platform. Other batch-oriented offerings like Hadoop include Nokia's Disco project and LexisNexis HPCC.
- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently.

Discuss

- Check out developerWorks blogs and get involved in the developerWorks community.
- Get involved in the developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

M. Tim Jones

M. Tim Jones is an embedded firmware architect and the author of Artificial Intelligence: A Systems Approach, GNU/Linux Application Programming (now in its second edition), AI Application Programming (in its second edition), and BSD Sockets Programming from a Multilanguage Perspective. His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim is a platform architect with Intel and author in Longmont, Colo.

© Copyright IBM Corporation 2012
(www.ibm.com/legal/copytrade.shtml)
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)